MarFS-campaign-lazy-fsio

One must figure out how to hide the gpfs metadata only mountpoint from the user, so it is possible that the lazy-fsio process family needs to do a private mount of the gpfs metadata name space using unshare or other mechanisms

This is a batch process that can walk either all of a file system or all of a tree. It initially is to count bytes in the tree and update a file at the mountpoint or high in the tree in a known place. It could also count files and build other useful info about the file sytem(part of the file system). This could be accomplished by running a stripped down version of pftool walker or running ilm query. This is primarily to use to compare with quoto for writing. It would also be used for things like a DF command for fuse etc. it should run quickly if possible and so you could run once a day or hour or something. It might be user requestable perhaps? It would be nice to see if GPFS has a way to tell you if large parts of the tree have changed or not to know if walking the entire tree is needed. There might be ways to do this.

This might be a reason to split the name space up and have a separate one for each campaign so you get some real parallelism if you are using ilm, etc. Over time the info collected could be richer .

You get place to store df info (and more) from config file fsinfo path variable

Minimum info you should probably put in that file.

Bytes of space used logically, bytes of space in trash, names, names in trash, current quota bytes and quota names you get quota from config file

* Periodically using the GPFS inode scan, scanning each GPFS file system in parallel to create information needed for statvfs/statfs and quota mgmt. as well as other useful histogram info from a simple and fast inode scan, files/space in histograms, histograms on file sizes, maybe also on dates (created in the last X days) modified in the last X days, maybe type histogram multi, uni, striped, packed, normal posix file, directory, maybe combos – why do all this, well its just so cheap, you reading the data in, keeping counters and adding stuff as you go is just amazingly cheap and it helps the sysadmins,
* You may want to exclude the trash directory from the space used calculation, perhaps by walking the trash dir and subtracting that info from the total for the file system
* We have both bytes used logical and bytes used physical
  + Logical is added up file sizes
  + Physical is added up MARpost xattr in field spaceused

So you may want to add up both values and use which ever for quotas

At a minimum, the namespace info file must have enough information to derive statvfs structure which is used by commands like df etc. in fuse (but you will want a lot more information in this file I suspect)

The statvfs structure looks like ths

struct statvfs {

unsigned long f\_bsize; /\* file system block size \*/

unsigned long f\_frsize; /\* fragment size \*/

fsblkcnt\_t f\_blocks; /\* size of fs in f\_frsize units \*/ fsblkcnt\_t f\_bfree; /\* # free blocks \*/

fsblkcnt\_t f\_bavail; /\* # free blocks for unprivileged users \*/

fsfilcnt\_t f\_files; /\* # inodes \*/

fsfilcnt\_t f\_ffree; /\* # free inodes \*/

fsfilcnt\_t f\_favail; /\* # free inodes for unprivileged users \*/ unsigned long f\_fsid; /\* file system ID \*/

unsigned long f\_flag; /\* mount flags \*/

unsigned long f\_namemax; /\* maximum filename length \*/ };

fs block size is probably just made up as is fragment size, size of units, fsid – although you could be creative here if you wanted, moutn flags, could be creative and max filename length is whatever your metadata file system allows – you could get that from statvfs.

Mostly you need to use the namespace info file to provide free space and inodes for quota use.